Absoft MacFortran 2.4

We continue to support the old MacFortran compiler in recognition of the fact that many of our original customers were MacFortran users. Users of MacFortran 2.4 should, however, be thinking about upgrading to Absoft's MPW MacFortran compiler since it contains many enhancements and is clearly the compiler that is getting most of Absoft's attention.

Example Programs The example programs have neither "f77.rl" nor any subroutines linked to them. They were compiled using MacFortran 2.4 without the 68020/68881 options set so that they would run on all Macs.

Include Files Unlike C and Pascal, standard Fortran does not support "record types", making it cumbersome to work with multiple "instances" of shared records. The closest approximation to a shared global record is a Fortran common block which has a set of variables equivalenced to it. An example of such a block can be found in the "FaceStorMF.inc" file or the corresponding file accompanying other FaceWare modules (its name will contain the substring "StorMF"). "FaceStorMF.inc" declares parameter names of core FaceWare commands and establishes a common block that contains an array named "fRec" and equivalenced variables that are shared with the core FaceWare modules. To make use of these variables, the following line should be inserted just after your main program variable declarations, but before the first executable statement:

```
include FaceStorMF.inc
```

All program units which directly access elements of the "fRec" shared record should also contain the above statement. Since numerous include statements can slow down compilation, you may want to create a smaller version of "FaceStorMF.inc" which contains only those equivalence statements which refer to elements of the storage array which you reference in your program, or simply make the necessary common block declaration and equivalence statements explicitly in your program code.

The file "FaceProcMF.inc" contains routines which are required for jumping to FaceWare modules. To make these routines available to your program, insert the following statement as the last line of your program file (and of any files used to create stand-alone, unlinked subroutines):

```
include FaceProcMF.inc
```

Writing To and Reading From Unit 9 (the Screen) when using FaceIt
If you take a look at the contents of "FaceProcMF.inc", you'll see that the default Fortran window, if present, is simply hidden at startup since the update events that it produces can cause problems for FaceIt. Fortunately, when MacFortran writes to and reads from the "screen" (unit 9) it makes use of whatever window is the current port. Since FaceIt always resets the current port to the active window, I/O directed to unit 9 appears in that window. You can change this to any other window (a modeless dialog would be a good choice) by simply resetting the current port using the toolbox call SETPORT before writing to unit 9. Note that the text written in this manner using unit 9 will not be editable, even if written to an editor window (a bad idea). The text is simply being drawn on the screen, with no connection to the text-editing module.

Using the MacFortran 2.4 Debugger with FaceIt After starting an application via the Debugger, set breakpoints within your source where you wish to examine the operation of your code. Then choose "Proceed to Breakpoint" to run your program to the first breakpoint. This may, of course, also require choosing an item from one of your menus if the code in question is only executed in response to a menu command. When a breakpoint is encountered, the Mac will beep and the Debugger source window will be brought to the front. If there is only one breakpoint set in your code, do not choose "Proceed to Breakpoint" again without doing at least one "Single Step". The Debugger loses track of a

breakpoint if asked to proceed in a loop from a breakpoint back to the same breakpoint.

While in the Debugger, the Debugger's keyboard shortcuts may conflict with keyboard commands which appear in the FaceIt menus or your program menus, in which case you may simply have to use the mouse to choose the affected menu item. Finally, the Debugger itself requires additional stack space, meaning that you may need to increase the stack space for your program to get it to run properly under the Debugger.

Other Programming Tips

- When practical, use "implicit none".

- Both source and data files should end with a carriage return.

- A common mistake when making "toolbx" calls is to forget to define the value of the integer constant used as the toolbox procedure name. Another mistake is to forget to declare "toolbx" as integer*4 when "toolbx" is being used as a function.

- When FaceIt is first called, you have the option of having it display a "deck of cards" cursor whenever your program has control. This is very useful when, for example, a Fortran error occurs within your program since you immediately know, by the form of the cursor, that the error is somewhere in your code and not within FaceIt. When such a Fortran error occurs, pressing the return key will usually take you back to the Finder.

- If using Select Case, you should be aware of the fact that the current version of MacFortran limits you to 32K of code per Select Case block.

- The compiler will not catch a mistaken number of arguments passed to a subroutine. For example, "call FaceIt(0,DoInit,0,0,0)" [5 arguments rather than 6] would get by the compiler, but will lead to unpredictable behavior of your program (probably a crash). So count those arguments!

- If your program works under some startup conditions, or on some Macs, but not others, then look first for an uninitialized variable. Variables are not zeroed at launch time, and will acquire the value of whatever garbage is sitting in memory unless you take the trouble to initialize them.

- MacFortran does not guarantee that strings declared in subroutines will be word-aligned on the stack (required by toolbox calls which accept strings as arguments). If you use such strings in toolbox calls, you'll get strange behavior like that seen with uninitialized variables. The workaround is to always use global string variables as parameters in toolbox calls (strings declared in your main program unit). The "uString" and "uName" variables, for example, are global, and can be used by your program for its own purposes, one of which may be to ensure that strings in toolbox calls are word-aligned. The most likely place that you'll be using such toolbox calls is in plotting routines that make calls to "STRINGWIDTH" and "DRAWSTRING".

- The toolbox call "FLUSHEVENTS" is incorrectly documented in the MacFortran include files. For an example of its proper use, see the include file "FaceProcMF.inc".

- If using Edit to edit your programs, you can mistakenly type the "Enter" key which inserts an invisible character that will later cause problems with the MacFortran compiler. Use Edit's "Show Invisibles" option to find and eliminate such characters.

- MacFortran does not allow you to pass integer constants to integer*2 subroutine arguments, so use integer*4 when passing constants.

- When passing variables to subroutines, the variables MUST be redeclared within the subroutines or they will be given undefined values. This error is NOT caught by the compiler, even if you have taken the trouble to use "implicit none" in the subroutine.

- When calling unlinked, stand-alone subroutines you may find that MacFortran will lose track of the location of such routines if the user, via a File menu command, has changed the default volume by opening or saving files. The solution (short of linking) is for you to reset the default volume before calling such subroutines. (You can also try putting the subroutines in the active System Folder, but this is not foolproof.) You can do this via the UtilIt command "SetRef".

- The linking and resource movement required to create a "stand-alone" application are outlined below. You can do some or all of the steps outlined, but they must be done in this order.

1. recompile program w/o "load" commands for routines to be linked
2. in the following order, link some or all of: program-specific subroutines (if any), "JumpIt.sub", "toolbx.sub", and either "f77.rl" or "m81.rl" (use "y" linker option to get the runtime library linked)
3. use MoveIt to move FaceWare resources to program file
4. use ResEdit to add the resources from your resource file